Cooperative Task Scheduler

Embedded Systems Specialization Program

CONTENT

- Main Features of presented Task Scheduler
  - Hardware Resources
  - Time-triggered tasks
  - Activation Algorithm (binary progression)
- Redesign into Cooperative Task Scheduler
  - Task States Model vs Task Scheduler functions
  - Tasks Priorities
  - How to implement a Schedule Point
  - How to convert external hardware events into Tasks
- Wrapping
- Activation and Execution

Embedded Systems

Main Features of Task Schedule

Main Features of Task Schedule

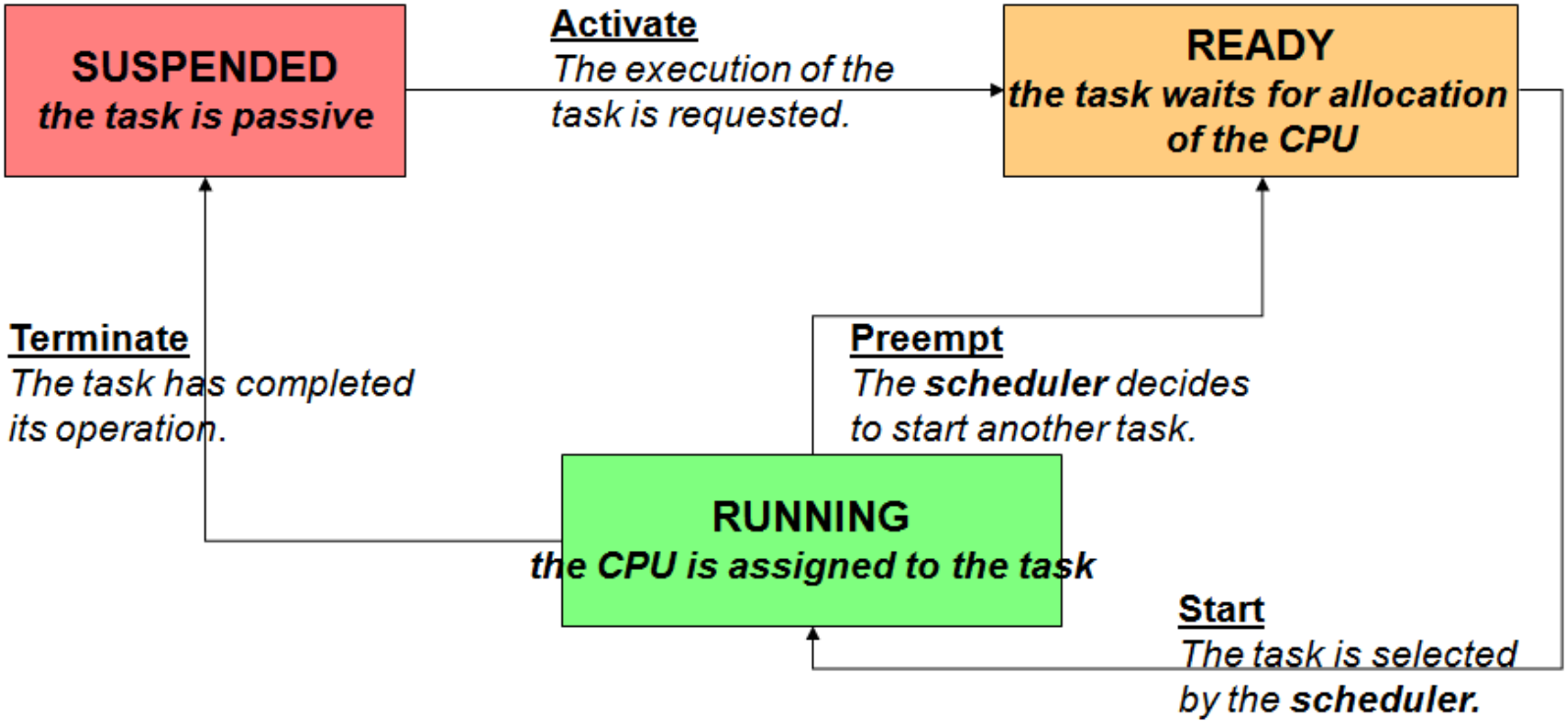
- Fast and stable non-preemptive Task Scheduler.
  - Based on periodic activation of Time-triggered tasks.
- Hardware Resources:
  - 1 hardware timer channel required (SysTick).

- Task Activation Algorithm:
  - Binary-progression Task Activation Algorithm allows for:
    - Fast tasks switching.
    - Balancing of CPU loading.
    - Avoiding concurrent task execution to a minimum.
  - Refer to “Task scheduler design.xls” excel file for details.

Main Features of Task Schedule

- Time Triggered Tasks:
  - 1ms, and two independent 2ms non-concurrent tasks.
  - 10ms, 50 ms and 100ms tasks.
  - Maximum of 500us (configurable) execution time for nonconcurrent tasks.

Task State Model



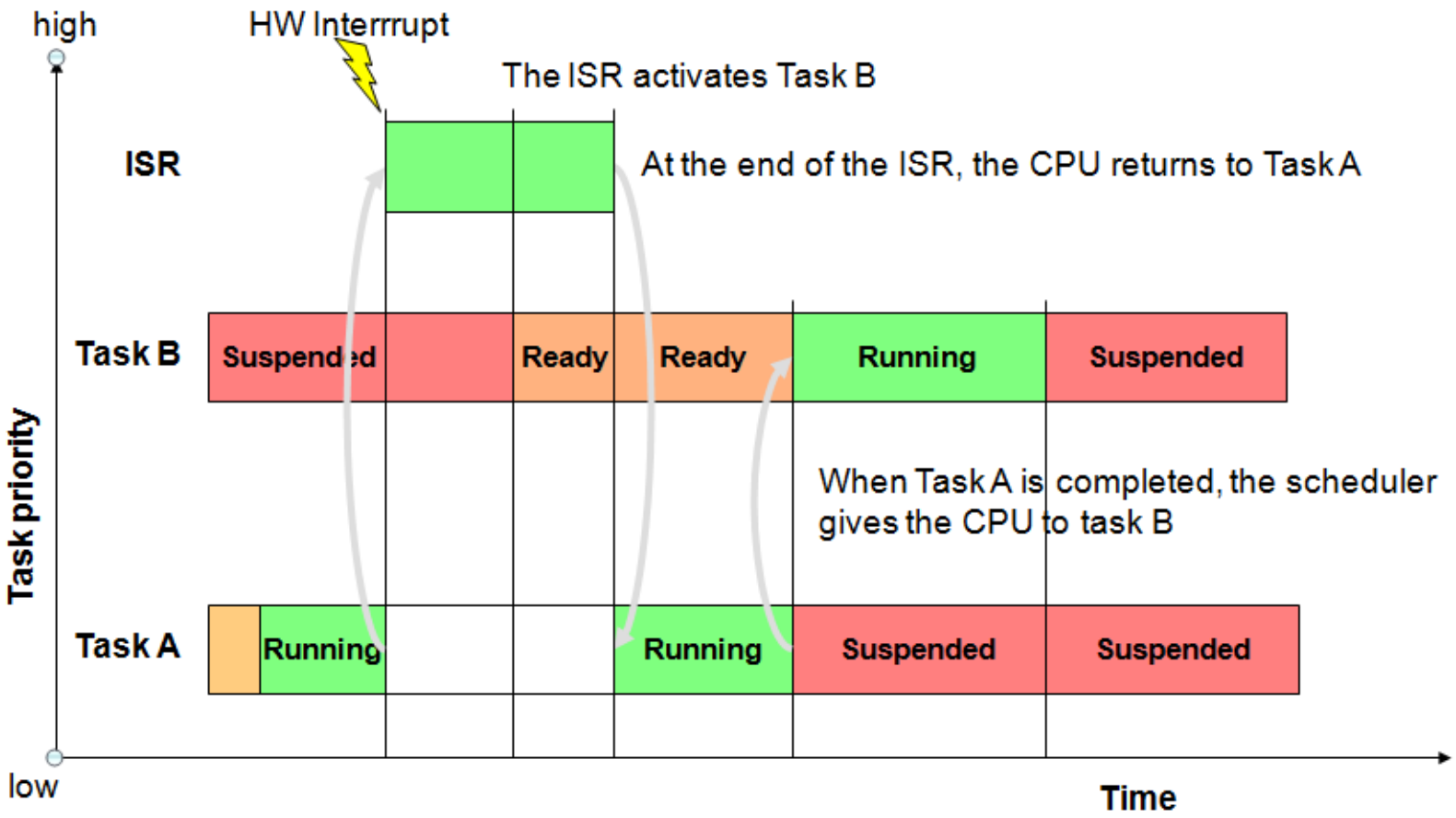
Embedded Systems

Preemptive Task Scheduling

Cooperative Task Scheduling

- Higher priority tasks may interrupt lower priority tasks.
- Each tasks runs until no longer has anything to process or until the scheduler decides to run a higher priority task.
- The implication is that a task can be interrupted (preempted) at any point by another task. When a task is preempted (i.e. interrupted by another task), its local context (variables, instruction pointer) have to be saved.
- The scheduler gives the CPU to a task according to its priority.
- The running task can be interrupted by an ISR (Interrupt Service Routine) or by a higher priority task.

Non-Preemptive Task Scheduling



Embedded Systems

Preemptive Task Scheduling

Preemptive Task Scheduling

- Higher priority tasks may interrupt lower priority tasks.
- Each tasks runs until no longer has anything to process or until the scheduler decides to run a higher priority task.
- The implication is that a task can be interrupted (preempted) at any point by another task. When a task is preempted (i.e. interrupted by another task), its local context (variables, instruction pointer) have to be saved.
- The scheduler gives the CPU to a task according to its priority.
- The running task can be interrupted by an ISR (Interrupt Service Routine) or by a higher priority task.

Embedded Systems

Redesign into Cooperative Task Scheduler

Redesign into Cooperative Task Scheduler

- Embedded States vs Task State Model:
  - Activate -> Task Scheduler
  - Start and Terminate -> Scheduler Callback

- Tasks Priorities
  - Assign to each task a given priority based on a pre-defined scheme:
    - 1ms -> 5 (highest)
    - 2ms_a and 2ms_b -> 4
    - 10ms -> 3
    - 50ms -> 2
    - 100ms -> 1 (lowest)

Redesign into Cooperative Task Scheduler

- How to implement a Schedule Point :
  - Redirect all relevant hardware events to a handler.
  - Handler will invoke a Task Scheduler Activation handler providing the address of the function to be called upon execution of the task and the task priority.
  - Task Scheduler handler will then activate task.
  - Execution of event-driven task will only be possible upon execution of a Schedule point and assuming the event-driven task has higher priority than task being currently executed.
  - After execution of event-driver task, scheduler will resume its normal operation from the Schedule point.
  - A Schedule point is simply another function with the following prototype:
    - void SchM_SchedulePoint(void);
  - Inside, it goes through the list of event-driven activated tasks (Status -> Ready)
  - If priority of these tasks is higher than priority of the currently executed task, then it proceeds to execution of tasks and removes them from the list.
  - Otherwise, event-driven tasks remain in Ready Status.